Finite State Automata and Simple Conceptual Graphs with Binary Conceptual Relations

Galia Angelova and Stoyan Mihov

Institute for Parallel Processing, Bulgarian Academy of Sciences
25A Acad. G. Bonchev Str., 1113 Sofia, Bulgaria
{galia, stoyan}@lml.bas.bg

Abstract. We propose a representation of simple conceptual graphs with binary conceptual relations, which is based on finite-state automata. The representation enables the calculation of injective projection as a two-stage process: *off-line* calculation of the computationally-intensive subsumption checks and encoding of the results as a minimal finite-state automaton, and *run-time* calculation by look-up in the minimal finite-state automaton using the projection query as a word belonging to the automaton language. This approach is feasible since a large part of the projection calculations does not depend on the run-time query but only on the relatively static statements in the knowledge base.

Keywords: projection, off-line preprocessing, efficient run-time calculations

1 Introduction

Conceptual Graphs (CGs) are based on logic and graph theory [1]. Many researchers contributed to the CGs elaboration and extension, e.g. the notion of support was formally introduced in a later paper [2]. The CG graphical structure visualises the identity of the variables, constants and predicate arguments in the corresponding logical formulas. A labeled graph morphism, called projection, defines specialisation and generalisation relations over CGs. Given two CGs G and H it holds that $H \leq G$ iff there is a projection from G to H [1,2]. The injective projection is an isomorphism, i.e. the image of G in H is a subgraph G^p of H such that G^p is isomorphic to G.

The most effective algorithms for computing CG projection rely on graph theory. They search for structural similarity and subgraph mappings between the projection query and the CGs in the Knowledge Base (KB). Given two CGs G and H, it is NP-complete to decide whether $G \geq H$. However there are large classes of CGs for which polynomial algorithms for projection exist when the underlying ordinary graphs are trees [3,4]. Projection is computed for the so-called Simple Conceptual Graphs (SCGs), which are equivalent to the positive, conjunctive and existential fragment of first order logic without functions [5]. Here we model the SCGs by finite-state automata (FSA), to exploit their operational efficiency.

Section 2 defines important notions. The FSA-based encoding of SCGs is presented in section 3. Section 4 sketches the idea of injective projection calculation in run-time. Experimental evaluations and the conclusion are given in section 5.

2 Basic Notions

Here we define the support for binary conceptual relations only:

Definition 1. A **support** S is a 4-tuple (T_C, T_R, I, τ) where:

- T_C is finite, partially ordered set of distinct concept types. For $x,y \in T_C$, $x \leq y$ means that x is a subtype of y; we say that x is a specialisation of y;
- T_R is finite, partially ordered set of distinct relation types. $T_C \cap T_R = \emptyset$. Each type $R \in T_R$ has arity 2 and holds either between two different concept types $x,y \in T_C$ or between two distinct instances of a concept type $x \in T_C$. Pairs $(c1_{maxR}, c2_{maxR}) \in T_C \times T_C$, called *star graphs*, are associated to each type $R \in T_R$; they define the greatest concept types that might be linked by R. A type $R \in T_R$ holds between $x,y \in T_C$ iff $x \leq c1_{maxR}$ and $y \leq c2_{maxR}$. For $R_1, R_2 \in T_R$ and $R_1 \leq R_2$, it holds that $c1_{maxR1} \leq c1_{maxR2}$ and $c2_{maxR1} \leq c2_{maxR2}$;
- I is a finite set of distinct individual markers (*referents*) denoting specified concept instances. $T_C \cap I = \emptyset$ and $T_R \cap I = \emptyset$. The *generic marker* $*$, $* \notin (T_C \cup T_R \cup I)$, refers to an unspecified individual of a specified concept type x. For all $i \in I$, $i \leq *$;
- The mapping τ: $I \rightarrow T_C$ defines correspondences between instances and concept types. So concept types have instances in contrast to the relations types. □

Definition 2. A **simple conceptual graph** with binary conceptual relations G, defined over a support S, is a connected, finite bipartite graph $(V = V_C \cup V_R, U, \lambda)$ where:

- The nodes V are defined by V_C – the set of concept nodes (c-nodes) and V_R – the set of relation nodes (r-nodes). $V_C \neq \emptyset$, i.e. each SCG contains at least one node;
- The edges U are defined by ordered pairs (x,r) or (r,y), where $x,y \in V_C$ and $r \in V_R$. The edges are directed either from a c-node to a r-node – like the *incoming arc* (x,r), or from a r-node to a c-node – like the *outgoing arc* (r,y). For every $r \in V_R$, there is exactly one incoming and one outgoing arcs, incident with r;
- The mapping λ associates labels of S to the elements of $V_C \cup V_R$. Each $c \in V_C$ is labeled by a pair $(C, marker(C))$, where $C \in T_C$ and $marker(C) \in I \cup \{*\}$. A c-node with generic marker is called a *generic node*, it refers to an unspecified individual of the specified concept type. A c-node with individual marker is called an *individual node*, it refers to a specified instance of the concept type. Each $r \in V_R$ is labeled by a type $R \in T_R$. The 1st argument of R is mapped to the c-node linked to the incoming arc of r while its 2nd argument is mapped to the c-node linked to the outgoing arc of r. □

The SCGs introduced by definition 2 can contain cycles but no multi-edges and loops. They may contain nodes with duplicating labels since λ associates repeating labels to the elements of $V_C \cup V_R$. Then all generic concept nodes of the same type are treated as distinct c-nodes of the underlying graph. Such nodes represent distinct instances, as we consider no coreference links between the c-nodes. We shall deal with *non-blank*, *simplified* SCGs [1] in *normal form* [5]. So we work with SCGs whose logical semantics is expressed by a 'minimal number' of support symbols. This is a kind of ′canonical′ format with exactly one label for each concept instance in the SCG logical formula and for each relation holding between two different instances.

Definition 3. A **injective projection** π: $G \rightarrow H$ is a graph morphism such that $\pi G \subseteq H$ has the properties: *(i)* for each concept c in G, πc is a concept in πG where $type(\pi c) \leq type(c)$. If c is individual, then $referent(c) = referent(\pi c)$; *(ii)* for each relation $r(c_1,c_2)$ in G, it holds that $\pi r(\pi c_1, \pi c_2)$ is in πG; *(iii)* πG is isomorphic to G. □

Definition 4. A **Finite State Automaton** A is a 5-tuple $\langle \Sigma, Q, q_0, F, \Delta \rangle$, where Σ is a finite alphabet, Q is a finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is the set of final states, and $\Delta \subseteq Q \times \Sigma \times Q$ is the *transition* relation. The transition $\langle q, a, p \rangle \in \Delta$ *begins* at state q, *ends* at state p and has the *label* a. □

Definition 5. Let A be a FSA. A **path** c in A is a finite sequence of $k>0$ transitions: $c = \langle q_0, a_1, q_1 \rangle \langle q_1, a_2, q_2 \rangle \ldots \langle q_{k-1}, a_k, q_k \rangle$, where $\langle q_{i-1}, a_i, q_i \rangle \in \Delta$ for $i = 1,\ldots,k$. The integer k is called the *length* of c. The state q_0 is called *beginning* of c and q_k is called the *end* of c. The string $w = a_1 a_2 \ldots a_k$ is called the *label* of c. The null path of $q \in Q$ begins and ends in q with label ε, where ε is the empty symbol. □

Definition 6. Let $A = \langle \Sigma, Q, q_0, F, \Delta \rangle$ be a FSA. Let Σ^* be the set of all strings over the alphabet Σ, including the empty symbol ε. The **generalised transition relation** Δ^* is the smallest subset of $Q\times\Sigma^*\times Q$ with the following two closure properties: *(i)* For all $q \in Q$ we have $\langle q, \varepsilon, q \rangle \in \Delta^*$; *(ii)* For all $q_1, q_2, q_3 \in Q$ and $w \in \Sigma^*$, $a \in \Sigma$: if $\langle q_1, w, q_2 \rangle \in \Delta^*$ and $\langle q_2, a, q_3 \rangle \in \Delta$, then $\langle q_1, w \cdot a, q_3 \rangle \in \Delta^*$. □

Definition 7. The **formal language** $L(A)$ accepted by a FSA $A = \langle \Sigma, Q, q_0, F, \Delta \rangle$ is the set of all strings, which are labels of paths leading from the initial to a final state: $L(A) := \{ w \in \Sigma^* \mid \exists\, q \in F : \langle q_0, w, q \rangle \in \Delta^* \}$. These strings are called **words** of $L(A)$. □

The FSAs accept regular languages. Every finite list of words over a finite alphabet of symbols is a regular language. Among the deterministic FSAs which accept a given language, there is a unique automaton (excluding isomorphisms) which has a minimal number of states [6]; it is called the **minimal** automaton of the language. There are algorithms which construct the minimal automata, given deterministic FSA.

Definition 8. Let $A = \langle \Sigma, Q, q_0, F, \Delta \rangle$ be a FSA. Let Σ^+ be the set of all strings w over the alphabet Σ, where $|w| \geq 1$. The automaton A is called **acyclic** iff for all $q \in Q$ there exist no string $w \in \Sigma^+$ such that $\langle q, w, q \rangle \in \Delta^*$. □

Definition 9. A **FSA with markers at the final states** A is a 7-tuple $\langle \Sigma, Q, q_0, F, \Delta, E, \mu \rangle$, where Σ is finite alphabet, Q is finite set of states, $q_0 \in Q$ is the initial state, $F \subseteq Q$ is set of final states, $\Delta \subseteq Q\times\Sigma\times Q$ is the *transition* relation, E is finite set of markers, and $\mu: F \rightarrow E$ is a function assigning a marker to each final state. □

3 Off-line Encoding of SCGs as Finite State Automata

We are looking for an internal encoding of the SCGs with binary conceptual relations, which maps the SCGs to words of symbols and provides a unified enumeration of: *(i)* all SCGs, defined according to a support, *(ii)* all their subgraphs and *(iii)* all injective generalisations of *(i)* and *(ii)*. Actually we aim to interpret them as a a finite regular language over certain finite alphabet. The encoding has to reflect the particular topological structure of the SCGs but should not contain graph-dependent indices, since we intend to further use this conceptual resource in run-time, when its symbols have to be matched to the symbols of (all future) projection queries. Perhaps all the subgraphs and their injective generalisations are too many and the brute-force enumeration makes no sense even if it is calculated off-line. However, we can filter only the subgraphs which have conceptual interpretation according to the support.

Definition 10. Let G be a SCG with binary conceptual relations. A **conceptual subgraph** $G_{cs} \subseteq G$ is a connected graph which is a SCG according to definition 2. □

Example 1. Figure 1 introduces a sample support, using examples from [1] and [4]:

T_C = {ATTRIBUTE, STATE, EVENT, ENTITY, ACT, PHYS-OBJECT, NAÏVE, LOVE, EGOISTIC, ANIMATE, ANIMAL, PERSON} with partial order shown at Fig. 1;

T_R = {ATTR, EXPR, OBJ } with partial order and star graphs shown at Fig. 1;

I = {John, Mary} which are not ordered; τ(John) = PERSON, τ(Mary) = PERSON.

Fig. 2 shows the SCGs G_1 and G_2, defined over the support given at Fig. 1. Figure 3A presents a conceptual subgraph of G_2. Fig. 3B shows a connected subset of G_2 nodes and edges, which has no conceptual interpretation according to the support. There exist connected bipartite graphs which cannot be interpreted as SCGs in any support, e.g. the one at Fig. 3B. Below by 'subgraphs' we shall mean 'conceptual subgraphs'.

The formula operator φ, defined in [1], translates non-blank SCGs with binary conceptual relations to logical formulas with monadic predicates, corresponding to the *c*-nodes, and binary predicates $rel(c_1,c_2)$ corresponding to the *r*-nodes. In the binary predicates, *rel* is a *r*-node label and c_1,c_2 are either variables for the generic *c*-nodes, or *referents* for the individual *c*-nodes. Replacing the variables by their *c*-nodes' labels and the referents by the string *type:referent*, where *type* is the label of the respective *c*-node in T_C, we can encode the information of the monadic predicates within the binary ones. Then every SCG formula $rel_1(c_{11},c_{12})$ & …. & $rel_k(c_{k1},c_{k2})$ can be easily linearised as a sequence of triples which consist of support symbols: c_{11} rel_1 c_{12} c_{21} rel_2 c_{22} … c_{k1} rel_k c_{k2} where c_{ij}, $1 \le i \le k$, j=1,2 are either concept type labels or strings *type:referent*.

The symbols used in this encoding correspond directly to the support labels which are meaningful for all SCGs in the KB as well as for the potential run-time projection queries. However, some of the generic concept types' labels might be duplicated, due to two different reasons: *(i)* they represent equivalent instances whose configuration reflects the topological structure of the underlying connected bipartite graph; *(ii)* they

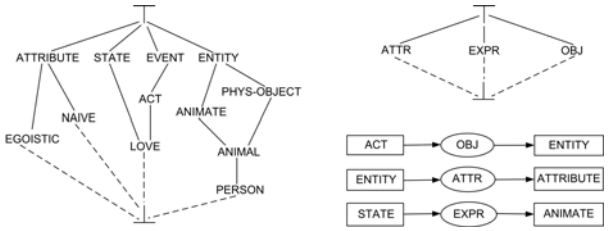

Figure 1. Partial order of concept and relation types and star graphs

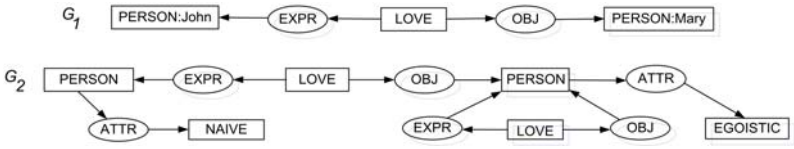

Figure 2. A knowledge base of two SCGs with binary conceptual relations G_1 and G_2

Figure 3A. A conceptual subgraph of G_2

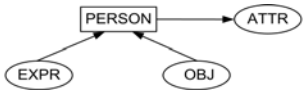

Figure 3B. Connected nodes of G_2, which do not form a conceptual subgraph. □

represent different unspecified instances belonging to the same concept type. Therefore we have to distinguish the two kinds of duplication and to mark the duplicated labels of the generic concept types, which refer to equivalent concept instances. For instance G_2 contains such duplications; the sequence of triple labels LOVE EXPR PERSON LOVE OBJ PERSON corresponds to three facts: *(f1)* "there exists a person who loves another person", *(f2)* "there exists a person who loves himself" and *(f3)* "there exist a person who experiences one kind of love and he/she is object of another love". A marker for the equivalences of the unspecified instances will ensure proper SCG encoding and, in addition, proper run-time treatment. Please note that a projection query with labels e.g. LOVE EXPR ANIMATE LOVE OBJ ANIMATE has to be projected in run-time to only one of (*f1*), (*f2*) and (*f3*) depending on its *c*-nodes identity. Therefore we need a unified approach to encode and recognise the *c*-nodes identity for all SCGs.

Describing all possible identities of *n* arguments is connected to the task of finding all ways to partition a set of *n* elements into nonempty, disjoint subsets. Each partition defines an equivalence relation of its members. The number of partitions is given by the so-called Bell numbers B_1, B_2, ... We are interested in partitions of even number of elements, since the arguments of binary conjuncts are even numbers. Let us consider in more detail the similarity between the partition task for a set of four elements and our task to define structural patterns for argument identity of two binary predicates with four arguments. Let the four set elements be x_1, y_1, x_2, y_2 and the SCG with two binary predicates be correspondingly $rel_1(x_1, y_1)$ & $rel_2(x_2, y_2)$.

Table 1, column 1 lists the set partitions into disjoint equivalence classes. These classes are interpreted as encodings of the topological links in a SCG with two binary predicates. Each row of column 2 contains either a G_1-G_2 subgraph with arguments linked correspondingly to the class in column 1, or comments why there are no such subgraphs. There are 15 ways to partition a set of four elements into disjoint subsets:

- Partition № 1 is irrelevant to our considerations as it corresponds to four distinct arguments of the two binary predicates – but the SCGs are to be connected;
- Eight partitions (№ 2, 5, 8, 9, 10, 11, 14, 15) are irrelevant, as they correspond to loops in the underlying graph, which are not allowed by definition 2; and
- Six patterns (corresponding to partitions № 3, 4, 6, 7, 12, 13) are relevant and provide a typology for the encoding of the links between two SCG binary predicates.

Thus the linearised SCG labels and the respective labels' identity annotations:

LOVE EXPR PERSON LOVE OBJ PERSON 1=3 (i.e. $x_1=x_2$) (1)
LOVE EXPR PERSON LOVE OBJ PERSON 1=3|2=4 (i.e. $x_1=x_2$ and $y_1=y_2$) (2)
LOVE EXPR PERSON LOVE OBJ PERSON 2=4 (i.e. $y_1=y_2$) (3)

uniquely encode the G_2 subgraphs (*f1*), (*f2*) and (*f3*). Moreover, we can reconstruct the logical formulas of the corresponding SCGs, given (1), (2) or (3), by building monadic predicates and adding the variables and the respective existential quantifiers. Thus we have found the encoding we needed; it is based on insights stemming from both the logical and graphical CG nature. Now we can work with the linear sequences of support labels and the associated identity annotations, interpreting them as SCGs.

We present an algorithm for the construction of a minimal acyclic automaton with markers at the final states, which builds a list of all KB subgraphs and their respective injective generalisations. All language' words are constructed here; results from automata theory build further the FSA itself [7]. We need some types and functions:

Equivalence classes	Examples of subgraph labels and comments
1. $\{\{x_1\},\{y_1\},\{x_2\},\{y_2\}\}$	irrelevant: distinct arguments build disconnected SCGs
2. $\{\{x_1, y_1\}, \{x_2\}, \{y_2\}\}$	$x_1=y_1$ is impossible as no loops are allowed
3. $\{\{x_1,x_2\}, \{y_1\}, \{y_2\}\}$	G_1: LOVE EXPR PERSON:John LOVE OBJ PERSON:Mary
4. $\{\{x_1\}, \{y_1, x_2\}, \{y_2\}\}$	G_2: LOVE EXPR PERSON PERSON ATTR NAIVE
5. $\{\{x_1, y_1, x_2\}, \{y_2\}\}$	$x_1=y_1$ is impossible as no loops are allowed
6. $\{\{x_1, y_2\}, \{y_1\}, \{x_2\}\}$	G_2: PERSON ATTR EGOISTIC LOVE EXPR PERSON
7. $\{\{x_1\}, \{y_1, y_2\}, \{x_2\}\}$	(*f3*) LOVE EXPR PERSON LOVE OBJ PERSON Acyclic subgraph of G_2 with distinct 1st arguments of the two conjuncts and equivalent 2nd arguments
8. $\{\{x_1\}, \{y_1\}, \{x_2,y_2\}\}$	$x_2=y_2$ is impossible as no loops are allowed
9. $\{\{x_1, y_1, y_2\}, \{x_2\}\}$	$x_1=y_1$ is impossible as no loops are allowed
10. $\{\{x_1, y_1\}, \{x_2, y_2\}\}$	$x_1=y_1$, $x_2=y_2$ is impossible as no loops are allowed
11. $\{\{x_1, x_2, y_2\}, \{y_1\}\}$	$x_2=y_2$ is impossible as no loops are allowed
12. $\{\{x_1, x_2\}, \{y_1, y_2\}\}$	(*f2*) LOVE EXPR PERSON LOVE OBJ PERSON Cyclic subgraph of G_2 with equivalent 1st and equivalent 2nd arguments of the two conjuncts
13. $\{\{x_1, y_2\}, \{y_1, x_2\}\}$	no such example in the sample graphs
14. $\{\{x_1\}, \{y_1, x_2, y_2\}\}$	$x_2=y_2$ is impossible as no loops are allowed
15. $\{\{x_1, y_1, x_2, y_2\}\}$	$x_1=y_1=x_2=y_2$ is impossible as no loops are allowed

Table 1. Partitions of a 4-element set and corresponding patterns of identical SCG arguments

CHAR-types: *lin_labels*, *identity, new_lin_labels*;
Arrays of lists: *list_subgraphs*; *list_gen_graphs;*
Arrays: *words_markers*(CHAR, <CHAR,CHAR,CHAR>) and
sorted_words_markers(CHAR, {<CHAR,CHAR,CHAR>, ..., <CHAR,CHAR,CHAR>});

function <*identity*(*G*), *lin_labels*(*G*)> = **GRAPH_LINEARISATION**(*G, Σ*) where *G* is a SCG presented in logical/graphical format over an ordered alphabet *Σ*. Given *G*, this function returns the pair (*i*) *identity*(*G*) – a sorted marker for identity of concept instances and *(ii) lin_labels*(*G*) which contains the linear sequence of sorted *G* labels, where each binary predicate in *G* is presented as a triple *concept1-relation-concept2*. The function integrates interfaces between our encoding and the other CG formats; it simplifies and normalises the input graph *G* and translates it to the desired linearised form. The sorted *identity*-marker is a string enumerating the equivalent *c*-nodes; it contains digits, '=' and '|' as shown in the samples (1), (2) and (3) above.

function *list_gen_graphs* = **COMPUTE_INJ_GEN**(*G,* Σ_1, Σ_2, λ). This function returns the list of all injective generalisations written in alphabet Σ_2, for a given graph *G* written in alphabet Σ_1. The generalisations are calculated using the mapping λ, which defines how the symbols of Σ_1 are to be generalised by symbols of Σ_2.

function $new\_lin\_labels(G^{sub})$ = **ENSURE_PROJ_MAPPING**($lin\_labels(G^{sub})$, $identity(G^{sub})$, Σ_1, $lin\_labels(G^{gen})$, $identity(G^{gen})$, Σ_2, λ)

Given a linearised subgraph G^{sub}, written in the ordered alphabet Σ_1 and its injective generalisation G^{gen}, written in the ordered alphabet Σ_2, this function checks whether the order of *c*-nodes in the sorted string $lin\_labels(G^{gen})$ corresponds to the order of the respective specialised *c*-nodes in the sorted string $lin\_labels(G^{sub})$. The check is

done following the mapping λ, which defines how the symbols of Σ_1 are to be generalised by symbols of Σ_2. (Remember that G^{sub} and G^{gen} contain equal number of binary predicates, where the ones of G^{gen} generalise some respective predicates of G^{sub}). If the *c*-nodes order in *lin_labels*(G^{gen}) corresponds to the order of the respective specialised *c*-nodes in *lin_labels*(G^{sub}), *new_lin_labels*(G^{sub}) = *lin_labels*(G^{sub}). Otherwise, *lin_labels*(G^{sub}) is rearranged in such a way that the order of its nodes is aligned to the order of generalising nodes in G^{gen}. Let *lin_labels*(G^{gen}) be as follows:

$$c^{gen}_{11}\ rel^{gen}_{1}\ c^{gen}_{12}\quad c^{gen}_{21}\ rel^{gen}_{2}\ c^{gen}_{22}\ \ldots\ c^{gen}_{k1}\ rel^{gen}_{k}\ c^{gen}_{k2}$$

where c^{gen}_{ij}, $1\le i\le k$, j=1,2 are labels of *c*-nodes and rel^{gen}_{i}, $1\le i\le k$, are labels of *r*-nodes. Then *lin_labels*(G^{sub}) is turned to the sequence of labels

$$c^{sub}_{11}\ rel^{sub}_{1}\ c^{sub}_{12}\quad c^{sub}_{21}\ rel^{sub}_{2}\ c^{sub}_{22}\ \ldots\ c^{sub}_{k1}\ rel^{sub}_{k}\ c^{sub}_{k2}$$

where $c^{gen}_{ij} \ge c^{sub}_{ij}$ for $1\le i\le k$, j=1,2 and $rel^{gen}_{i} \ge rel^{sub}_{i}$ for $1\le i\le k$. The re-arranged labels of G^{sub} nodes are returned in *new_lin_labels*(G^{sub}). The string *new_lin_labels*(G^{sub}) is no longer lexicographically sorted but its nodes' order is aligned to the order of the generalising nodes in G^{gen}. The *c*-nodes' topological links in *new_lin_labels*(G^{sub}) are given by *identity*(G^{gen}). Thus an injective projection π: $G^{gen}\to G^{sub}$ is encoded.

Algorithm 1. *Construction of a minimal acyclic FSA with markers at the final states* $A_{KB} = \langle \Sigma, Q, q_0, F, \Delta, E, \mu \rangle$ *which encodes all subgraphs' injective generalisations for a KB of SCGs with binary conceptual relations* $\{G_1, G_2, \ldots, G_n\}$ *over support S.*

Step 1, defining the finite alphabet Σ: Let $S = (T_C, T_R, I, \tau)$ be the KB support according to definition 1. Define $\Sigma = \{x \mid x\in T_C \text{ or } x\in T_R\} \cup \{ x{:}i \mid x\in T_C,\ i\in I \text{ and } \tau(i)=x\}$. Order the m symbols of Σ using certain lexicographic order $\Omega = \langle a_1, a_2, \ldots, a_m \rangle$.

Step 2, indexing all c-nodes: Juxtapose distinct integer indices to all KB *c*-nodes, to ensure their default treatment as distinct instances of the generic concept types. Then $\Sigma_{KB} = \{a_{ij} \mid a_i\in\Sigma,\ 1\le i\le m$ and j is an index assigned to the KB *c*-node a_i, $1\le j\le p_i$ or j='*none*' when no indices are assigned to $a_i\}$.

Order the symbols of Σ_{KB} according to the lexicographic order

$\Omega_{KB} = \langle a_{1s1}, \ldots, a_{1su}, a_{2p1}, \ldots, a_{2pv}, \ldots.., a_{mq1}, \ldots, a_{mqx} \rangle$ where $s_1, s_2, \ldots s_u$ are the indices assigned to a_1; $p_1, \ldots, p_v$ are the indices assigned to a_2; $q_1, \ldots q_x$ are the indices assigned to a_m and $s_1 < s_2 < \ldots < s_u$, $p_1 < p_2 < \ldots < p_v$, ….. and $q_1 < q_2 < \ldots < q_x$.

Define a mapping λ: $\Sigma_{KB}\to\Sigma$ where $\lambda(a_{ij}) = a_i$ for each $a_{ij}\in\Sigma_{KB}$, $1\le i\le m$ and j is an index assigned in Σ_{KB} to the symbol $a_i\in\Sigma$.

/* Step 3, computation of all KB (conceptual) subgraphs: */

for $i := 1$ **to** n **do begin**

list_subgraphs(i) := { $G^{sub\text{-}j}{}_i \mid G^{sub\text{-}j}{}_i \subseteq G_i$ according to definition 10}; **end;**

/* Step 4, computation and encoding of all injective generalisations: */

var *gen_index* := 1;

for each i **and** $G^{sub\text{-}j}{}_i$ in *list_subgraphs(i)* **do begin**

<*identity*($G^{sub\text{-}j}{}_i$), *lin_labels*($G^{sub\text{-}j}{}_i$)> := GRAPH_LINEARISATION($G^{sub\text{-}j}{}_i$, Σ_{KB});

list_gen_graphs(i,j) := COMPUTE_INJ_GEN($G^{sub\text{-}j}{}_i$, Σ_{KB}, Σ, λ);

for each G^{gen} in *list_gen_graphs(i,j)* **do begin**

<*identity*(G^{gen}), *lin_labels*(G^{gen})> := GRAPH_LINEARISATION(G^{gen}, Σ);

new_lin_labels($G^{sub\text{-}j}{}_i$) := ENSURE_PROJ_MAPPING(*lin_labels*($G^{sub\text{-}j}{}_i$), *identity*($G^{sub\text{-}j}{}_i$), Σ_{KB}, *lin_labels*(G^{gen}), *identity*(G^{gen}), Σ, λ);

words_markers(*gen_index, 1*) := *lin_labels*(G^{gen});

words_markers(*gen_index, 2)* := <*identity*(G^{gen}), *new_lin_labels*($G^{sub\text{-}j}_i$), G_i>;
gen_index := *gen_index*+1; **end; end;**
sorted_words_markers := SORT-BY-FIRST-COLUMN(*words_markers*) ;
while *sorted_words_markers*(*,1) *contains k>1 repeating words in column 1, starting at row p* **do begin**
sorted_words_markers(*p, 2*) := {*sorted_words_markers*(*p*,2), *sorted_words_markers*(*p*+1,2),…, *sorted_words_markers*(*p*+*k*-1,2)};
for 1≤*s*≤*k*-1 **do begin** DELETE-ROW(*sorted_words_markers*(*p*+*s*,*) **end**; **end**;
$L = \{w_1, w_2, \ldots, w_z \mid w_i \in$ *sorted_words_markers*(*,1), $1 \le i \le z$ and $w_i \le w_j$ according to Ω, for $i \le j$, $1 \le i \le z$ and $1 \le j \le z$ $\}$.

/* *Step 5, FSA construction*: */

Consider L as a finite language over Σ, given as a list of words sorted according to Ω. Apply results of [7] and build directly the minimal acyclic FSA with markers at the final states $A_{KB}=\langle \Sigma, Q, q_0, F, \Delta, E, \mu \rangle$, which recognises $L=\{w_1,\ldots,w_z\}$. Then

$F=\{q_{wi} | q_{wi}$ is the end of the path beginning at q_0 with label w_i, for $w_i \in L$, $1 \le i \le z\}$.

$E=\{ M_i \mid M_i=$*sorted_words_markers*$(i,2)$, $1 \le i \le z\}$ and μ: $q_{wi} \rightarrow M_i$ where $q_{wi} \in F$, *sorted_words_markers*$(i,1) = w_i$ and *sorted_words_markers*$(i,2) = M_i$. □

Example 2. We list below 7 (out of 37) subgraphs of the KB at Fig. 2. They are given as markers <*identity-type*, *linear-subgraph-labels*, *index-of-main-KB-graph*>:

M_1: <none, LOVE EXPR PERSON:John, G_1>
M_2: <1=3, LOVE EXPR PERSON:John LOVE OBJ PERSON:Mary, G_1>
M_3: <none, LOVE EXPR PERSON, G_2>
M_4: <1=3, LOVE EXPR PERSON LOVE OBJ PERSON, G_2>
M_5: <1=3|2=4, LOVE EXPR PERSON LOVE OBJ PERSON, G_2>
M_6: <2=4, LOVE EXPR PERSON LOVE OBJ PERSON, G_2>
M_7: <1=3|2=5, LOVE EXPR PERSON LOVE OBJ PERSON PERSON ATTR NAIVE, G_2>

Fig. 4 shows the minimal FSA with markers at the final states, which encodes the 33 injective generalisations of the subgraphs in M_1-M_7. New markers M_8-M_{11} were created at step 4 of algorithm 1, to properly encode all data.

4 Injective Projection in Run-Time

The injective projection is calculated by a look-up in the minimal acyclic FSA, which encodes all the KB generalisations, with a word built by the query graph labels. There are two main on-line tasks, given a query G: *(i) Presenting G as a sorted sequence of support symbols*, and *calculation of its identity-type* for linear time O(n); *(ii) Look-up in the FSA A_{KB} by a word w_G*. Its complexity is clearly O(n), where n is the number of G symbols. No matter how large the KB is, all injective projections of G to the KB are found at once with complexity depending on the input length only.

Now we see the benefits of the suggested explicit off-line enumerations. Actually we enumerate all possible injective mappings from all injective projection queries to the KB subgraphs. It becomes trivial to check whether a SCG with binary conceptual relations is equivalent to certain SCG in the KB. Thus the lexicographic ordering of conceptual labels provides a convenient formal framework for SCGs comparison.

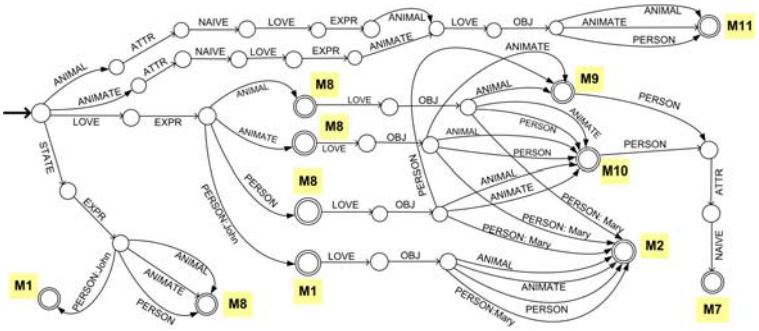

Figure 4. Minimal FSA, encoding all injective generalisations for 7 subgraphs of G_1 and G_2. □

5 Initial Experiments

We have generated randomly type hierarchies of 600 concept types and 40 relation types. The experimental KB consists of 291 SCGs with binary conceptual relations in normal form, each with length of 3-10 conjuncts. These SCGs have 6753 (conceptual) subgraphs with 10436190 different injective generalisations. After the lexicographic sorting of all words (injective generalisations' labels) is done, they belong to 13885 *identity*-types- i.e. they are topologically structured in a relatively uniform way. The minimal acyclic FSA with markers at the final states, which recognises all injective generalisations, has 2751977 states and 3972096 transition arcs. The input text file of sorted words, prepared for the FSA construction, is 891,4 MB. The minimal FSA is 52,44 MB but the markers-subgraphs are encoded externally, i.e. markers contains only pointers. The input text file is compressed about 18 times when building the minimal FSA, which is only 2,4 times bigger than the zipped version of the input file.

The suggested approach implements off-line as much computations as possible and provides exclusive run-time efficiency. The implementation requires considerable off-line preprocessing and large space since the off-line tasks operate on raw data. The star graphs impose strong constraints on the structural patterns while computing injective generalisations; this is intuitively clear but now we see experimental evidences about the ′uniformity′. Currently we plan an experiment with realistic data.

References

1. Sowa, J. *Conceptual Structures – Inform. Processing in Mind and Machine*. Reading, 1984.
2. Chein, M. and M.-L. Mugnier. *Conceptual Graphs: fundamental notions*. Revue d'Intelligence Artificielle, Vol. 6, no. 4, 1992, pp. 365-406.
3. Mugnier, M.-L. and M. Chein. *Polynomial Algorithms for Projection and Matching,* In: 7th Annual Workshop on Conceptual Graphs (AWCG'92), 1992, pp. 49-58.
4. Mugnier, M.-L. *On Generalization / Specialization for Conceptual Graphs*. Journal of Experimental and Theoretical Computer Science, Vol. 7, 1995, pp. 325-344.
5. Baget, J.-F. and M.-L. Mugnier. *Extensions of Simple Conceptual Graphs: the Complexity of Rules and Constraints*. JAIR, vol. 16, 2002, pp. 425-465.
6. Hopcroft, J. and J. Ullman. *Introduction to Automata Theory, Languages, and Computation.* Addison-Wesley, Reading, MA, 1979.
7. Daciuk, J., St. Mihov, B. Watson, and R. Watson, *Incremental Construction of Minimal Acyclic Finite State Automata,* J. of Comp. Linguistics, Vol. 26, Issue 1, 2000, pp. 3-16.